1. **Is it okay to initialize all the weights to the same value as long as that value is selected randomly using He initialization?**

**No. All weights should be initialized to different random values and should not have the same initial value. If weights are symmetrical, meaning they have the same value, it makes it almost impossible for backpropagation to converge to a good solution.**

**Think of it this way: if all the weights are the same, it's like having just one neuron per layer, but much slower.**

**The technique we use to break this symmetry is to sample weights randomly.**

1. **Is it okay to initialize the bias terms to 0?**

**Yes, this is fine. It does not end up making much of a difference. It's been my practice to initialize the bias term to 1.**

1. **Name three advantages of the ELU activation function over ReLU.**

* **It can take on negative values, so the average output of the neurons in any given layer is typically closer to 0 then when using the ReLU function. This helps alleviate the vanishing gradients problem. The vanishing gradients problem is the idea that gradients often get smaller and smaller as the algorithm progresses down to the lower layers. As a result, the gradient descent update leaves the lower layer connection weights virtually unshanged, and training never converges to a good solution.**
* **It always has a non-zero derivative, which avoids the dying units issue that can affect ReLU units. "Dying ReLU's" is the problem where units stop outputting anything other than 0. In some cases, you may find that half of your networks neurons are dead, especially if you use a large learning rate.**
* **It is smooth everywhere, which helps gradient descent converge faster. ReLU's slope abruptly jumps from 0 to 1 at z = 0. Such an abrupt change can slow down gradient descent because it will bounce around z = 0.**

1. **In which cases would you want to use each of the following activation functions: ELU, leaky ReLU (and its variants), ReLU, tanh, logistic, and softmax?**

**The ELU function is a good default.**

**If you need the network to be as fast as possible, you can use one of the leaky ReLU variants.**

**The basic ReLU function is also preferred due to its simplicity, despite the fact they are generally outperformed by ELU and leaky ReLU. If you have a situation where it's preferable to have neurons output exactly 0, then ReLU is a good choice.**

**tanh can be useful in the output layer if you need your outputs to be between -1 and 1. It's not used too much in hidden layers.**

**The logistic function is useful in the output layer when you need to estimate a probability like in the binary, multi-class, or multi-class multi-label classification problems. Like tanh, it is not used in the hidden layers.**

**The softmax function is useful in the output layer when you need to output probabilities for mutually exclusive classes. Again, it's not used in hidden layers.**

1. **What may happen if you set the momentum hyperparameter too close to 1 (e.g., 0.99999) when using a MomentumOptimizer?**

**The purpose of the momentum hyperparameter B is to simulate friction and prevent the momentum from growing too large. We set it to 0 for high friction, and 1 for no friction.**

**As the hyperparameter gets closer to 1, there will be less friction and the momentum optimization will "roll faster down the hill." This means that the optimizer will overshoot, then come back, overshoot again, and oscillate like this many times before stabilizing at the minimum. This is one of the reasons why it is good to have a bit of friction in the system: it gets rid of these oscillations and thus speeds up convergence.**

1. **Name three ways you can produce a sparse model.**

**A sparse model is that where most weights are equal to 0. There's a couple ways of achieving that effect.**

**You can train the model normally then zero out tiny weights.**

**For more sparsity, you can apply l1 regularization during training, which pushes the optimizer towards sparsity.**

**Finally, you can combine l1 regulatization with dual averaging using TensorFlow's FTRLOptimizer class.**

1. **Does dropout slow down training? Does it slow down inference (i.e., making predictions on new instances)?**

**Dropout is a popular regularization technique for deep neural networks.**

**The algorithm is: at each training step, every neuron (including the input neurons but excluding the output neurons) has a probability p of being temporariliy "dropped out", meaning it will be entierly ignored during this training step. However, it may be active during the next step.**

**The hyperparameter p is called the dropout rate, and it is typically set to 0.5.**

**After training the neurons don't get dropped anymore. That's the gist of the algorithm.**

**Dropout does tend to significantly slow down convergence, but it usually results in a much better model when tuned properly. Remember, because dropout is only tuned during training, it has no impact on inference.**